Python Code Examples

1 Basic - variable assignment and math operations

Name and age greeting

This script simply asks the user for their name and age and greets them. The script uses *f-strings* to include the input values in the printed statements.

```
name = input("What is your name? ")
age = int(input("What is your age? "))
print()    # empty print() creates a blank line
print(f"Hello, {name}")
print(f"Happy {age}!")
```

```
What is your name? Brian
What is your age? 55

Hello, Brian
Happy 55!
```

Ideal gas law – pressure

The ideal gas law is used quite a bit in Thermodynamics. This user-defined function uses $pV = mRT$ to solve for and return the pressure p in kPa.

```
def gas_law_pressure(m, R, T, V):
    """
    Calculate pressure using the ideal gas law
    Arguments: m = mass in kg
               R = gas constant for specific gas in kJ/(kg K)
               T = temperature in K
               V = volume in m^3
    Returns: p = pressure in kPa
    """
    return m*R*T/V

print(gas_law_pressure(0.2, 0.287, 325, 0.05))
```

```
373.1
```

Conical Frustum

The volume and surface area of a conical frustum, aka truncated cone, can be determined using the following expressions. The example script that follows calculates and prints both after asking the user for the two radii, the height, and the units of measure.

$$V = \frac{\pi h}{3}(r_1^2 + r_1 r_2 + r_2^2)$$

$$SA = \pi \left[(r_1 + r_2)\sqrt{(r_2 - r_1)^2 + h^2} + r_1^2 + r_2^2 \right]$$

```
import math
print("Conical Frustum\n")
units = input("Enter the units of measure: ")
h = float(input("Enter the height in {units}: "))
r1 = float(input(f"Enter the value radius 1 in {units}: "))
r2 = float(input(f"Enter the value radius 2 in {units}: "))

V = math.pi*h/3 * (r1**2 + r1*r2 + r2**2)
SA = math.pi*((r1 + r2)*math.sqrt((r2 - r2)**2 + h**2) + r1**2 + r2**2)

print()
print(f"The volume is {V:.2f} {units}\u00b3")
print(f"The surface area is {SA:.2f} {units}\u00b2")
```

```
Conical Frustum

Enter the units of measure: inch
Enter the height in {units}: 12
Enter the value radius 1 in inch: 6.2
Enter the value radius 2 in inch: 8.4

The volume is 2024.19 inch³
The surface area is 892.84 inch²
```

2 Conditional Branching – `if`, `if-else`, and `if-elif-else`

Branching example 1

3 Iteration – `for` and `while` loops

Fibonacci sequence – example 1

The Fibonacci numbers are a sequence of numbers in which the first two elements are 0 and 1 and the value of each subsequent element in the sequence is the sum of the previous two elements, i.e. $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

The following function called `fibonacci(x)` returns the first `x` Fibonacci numbers as a list using a `for` loop for iteration. It starts with a list containing 0 and 1 and appends each additional value until there are `x` values in the list.

```
def fibonacci(x):
    fib = [0, 1]   # start with 0 & 1 in a list of Fibonacci numbers
    # keep in mind that the list already has 2 values
    for i in range(x - 2):
        # append the list 'fib' with the sum of the last two values
        # in 'fib' using indexing from the right
        fib.append(fib[-1] + fib[-2])
    return fib
```

```
>>> fibonacci(10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

>>> fibonacci(20)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
↪  2584, 4181]

>>> fibonacci(30)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
↪  2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,
↪  317811, 514229]
```

Fibonacci sequence – example 2

Another option is to assign the last two values in the sequence to variables and change the variables each time through a loop. This method still requires appending to a list in order to return a list, as is desired. The first method that uses indexing is more efficient.

```
def fibonacci(x):
    a, b = 0, 1
    fib = [a, b]
    for i in range(x - 2):
        fib.append(a + b)
        # update a and b at the same time, where a = b and b = a + b
        a, b = b, a + b
    return fib
```

```
>>> fibonacci(10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

>>> fibonacci(20)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
↪  2584, 4181]

>>> fibonacci(30)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
↪  2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,
↪  317811, 514229]
```

Years to double an investment

The following function uses a `while` loop to determine the number of years required to at least double an investment. Each iteration of the loop will increment the number of years by one to keep track of how many have elapsed, even though the number of years is not used directly in the calculation. After one iteration the number of years will be equal to one.

```
def double_investment(amount, rate):
    """
    Calculates the number of years to double an investment using
    simple annual rate of return.
    Arguments: amount => starting investment
               rate => annual rate of return without % (4.5% is 4.5)
    Prints: Number of years to (at least) double and the final value
    """
    years = 0
    goal = amount * 2
    while amount < goal:
        amount = amount * (1 + rate/100)
        years += 1
    amount = round(amount, 2)
    print(f"It will take {years} years to double your investment"
          f" at a {rate}% annual rate")
    print(f"Your investment will be worth ${amount} after {years} years")
```

```
>>> double_investment(100, 5)
It will take 15 years to double your investment at a 5% annual rate
Your investment will be worth $207.89 after 15 years

>>> double_investment(2000, 7.2)
It will take 10 years to double your investment at a 7.2% annual rate
Your investment will be worth $4008.46 after 10 years
```

Largest factorial

The factorial $n!$ is the product of all integers from 1 to n where n is a positive integer. The following code block demonstrates a function that determines the largest factorial less than or equal to the user specified value `limit`. The printed results include the integer n and its factorial. A `while` loop is used whose condition "looks ahead" at the factorial of the next integer to determine if it is less than or equal to the limit.

```
def largest_factorial(limit):
    import math
    n = 1
    while math.factorial(n + 1) <= limit:
        n += 1
    print(f"{n}! is the largest factorial <= {limit:,d}: "
          f"{n}! = {math.factorial(n):,d}")
```

```
>>> largest_factorial(120)
5! is the largest factorial <= 120: 5! = 120

>>> largest_factorial(1000)
6! is the largest factorial <= 1,000: 6! = 720

>>> largest_factorial(10_000)
7! is the largest factorial <= 10,000: 7! = 5,040

>>> largest_factorial(100_000)
8! is the largest factorial <= 100,000: 8! = 40,320
```